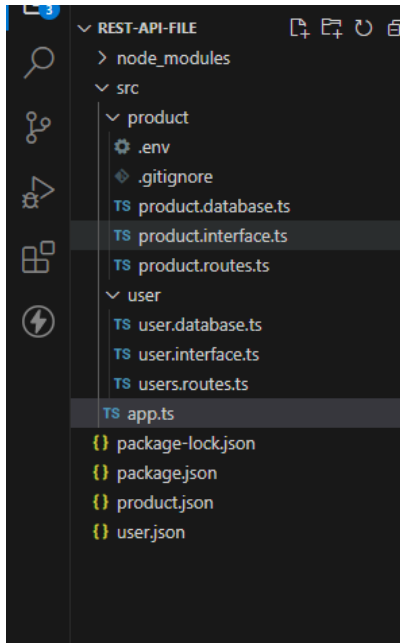


Eldrin A. Trapa
Activity 4

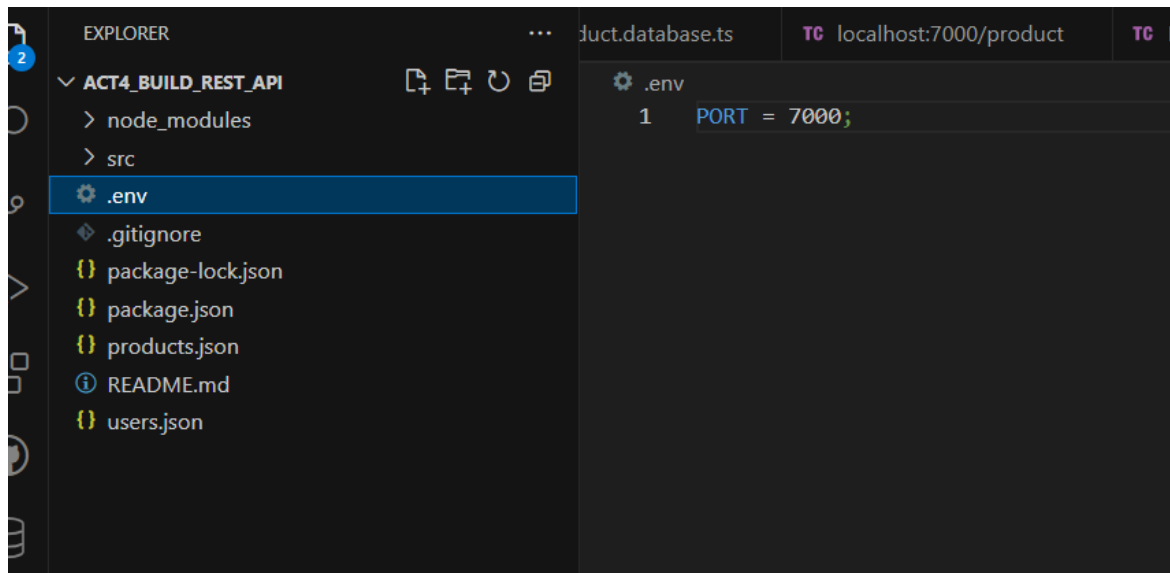
I start by creating a project directory



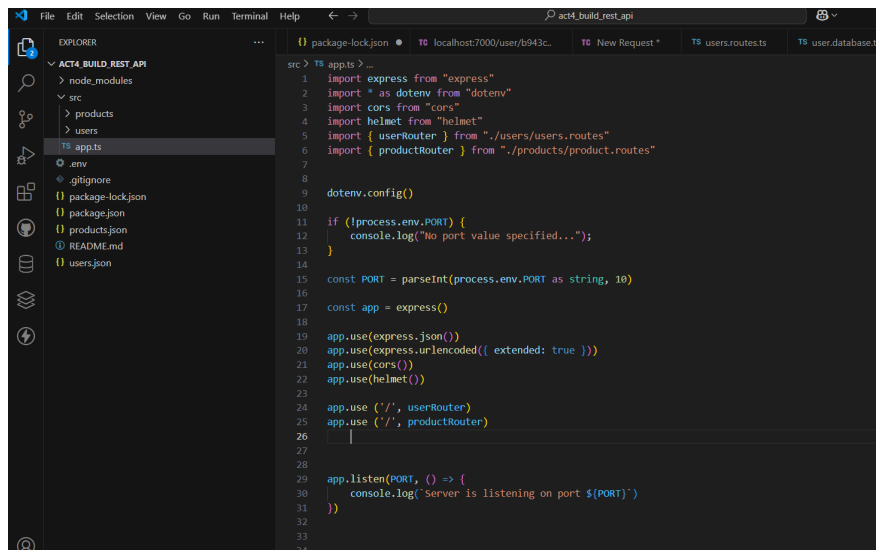
Then, i installed all needed package

```
C:\Windows\System32\cmd.e x + v
C:\Users\yy445\Documents\REST-API-FILE>npm install dotenv
up to date, audited 156 packages in 1s
24 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
C:\Users\yy445\Documents\REST-API-FILE>npm i -D ts-node-dev
up to date, audited 156 packages in 5s
24 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
C:\Users\yy445\Documents\REST-API-FILE>npm install --save-dev typescript @types/node @types/express ts-node
up to date, audited 156 packages in 7s
24 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
C:\Users\yy445\Documents\REST-API-FILE>
C:\Users\yy445\Documents\REST-API-FILE>
```

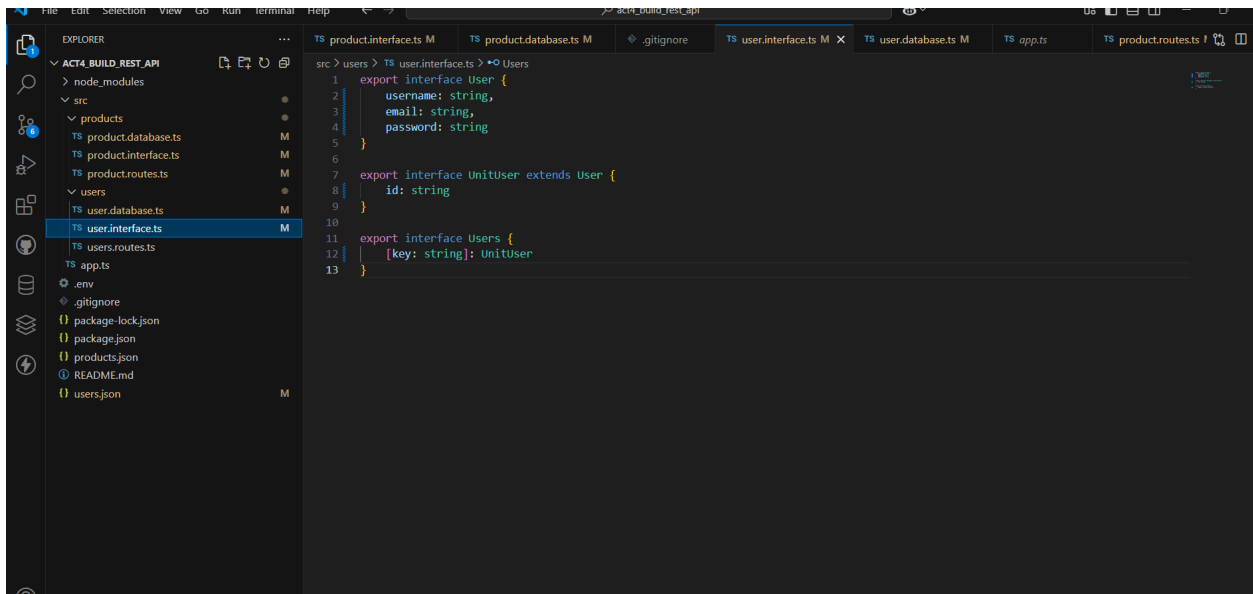
This is the .env with PORT = 7000



This is app.ts to locate the app.ts file in the root of the src folder and import the project dependencies



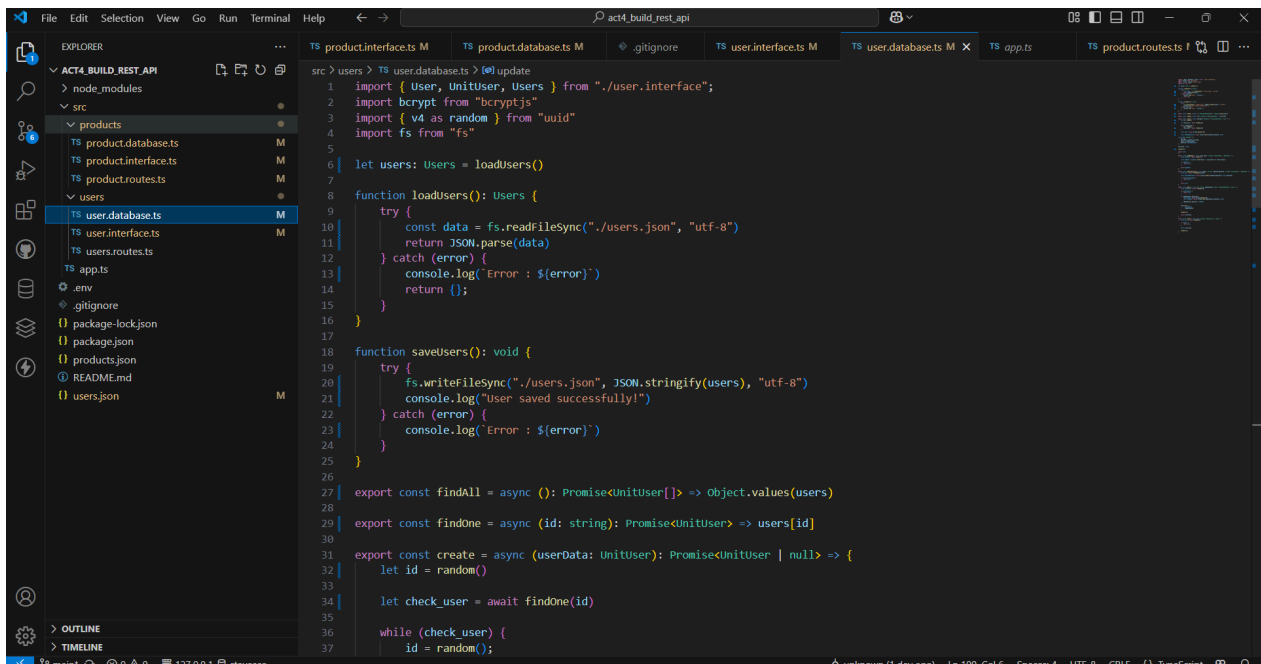
This is the user.interface **src/users/user.interface.ts**



Next is create the user.database for logic to our data storage with this

src/users/user.database.ts

And this is the following code:



This screenshot shows the implementation of a `create` function in the `src > users > TS user.databases.ts` file. The function is an asynchronous operation that takes a `UnitUser` object and returns a `Promise<UnitUser | null>`. It performs the following steps:

- Generates a random ID using `id = random()`.
- Checks if a user with that ID already exists using `await findOne(id)`.
- Generates a salt and hashes the password using `await bcrypt.genSalt(10)` and `await bcrypt.hash(userData.password, salt)`.
- Creates a new user object with the generated ID, username, email, and hashed password.
- Saves the new user to the database using `saveUsers()`.
- Returns the newly created user object.

Below the `create` function, there is a `findByEmail` function that takes an email and returns a `Promise<null | UnitUser>`. It finds all users and returns the one whose email matches the provided email.

```
src > users > TS user.databases.ts > create
31 export const create = async (userData: UnitUser): Promise<UnitUser | null> => {
32   while (checkUser) {
33     id = random();
34     check_user = await findOne(id)
35   }
36
37   const salt = await bcrypt.genSalt(10)
38
39   const hashedPassword = await bcrypt.hash(userData.password, salt)
40
41   const user: UnitUser = {
42     id: id,
43     username: userData.username,
44     email: userData.email,
45     password: hashedPassword
46   };
47   users[id] = user;
48
49   saveUsers()
50
51   return user;
52 };
53
54 export const findByEmail = async (user_email: string): Promise<null | UnitUser> => {
55   const allUsers = await findAll();
56
57   const getUser = allUsers.find(result => user_email === result.email);
58
59   if (!getUser) {
60     return null;
61   }
62   return getUser;
63 };
64
65 export const comparePassword = async (email: string, supplied_password: string): Promise<null | UnitUser> => {
```

This screenshot shows the implementation of `update` and `remove` functions in the `src > users > TS user.databases.ts` file.

The `update` function is an asynchronous operation that takes an `id`, a `string` for the email, and `updateValues` of type `User`. It returns a `Promise<UnitUser | null>`. The steps are:

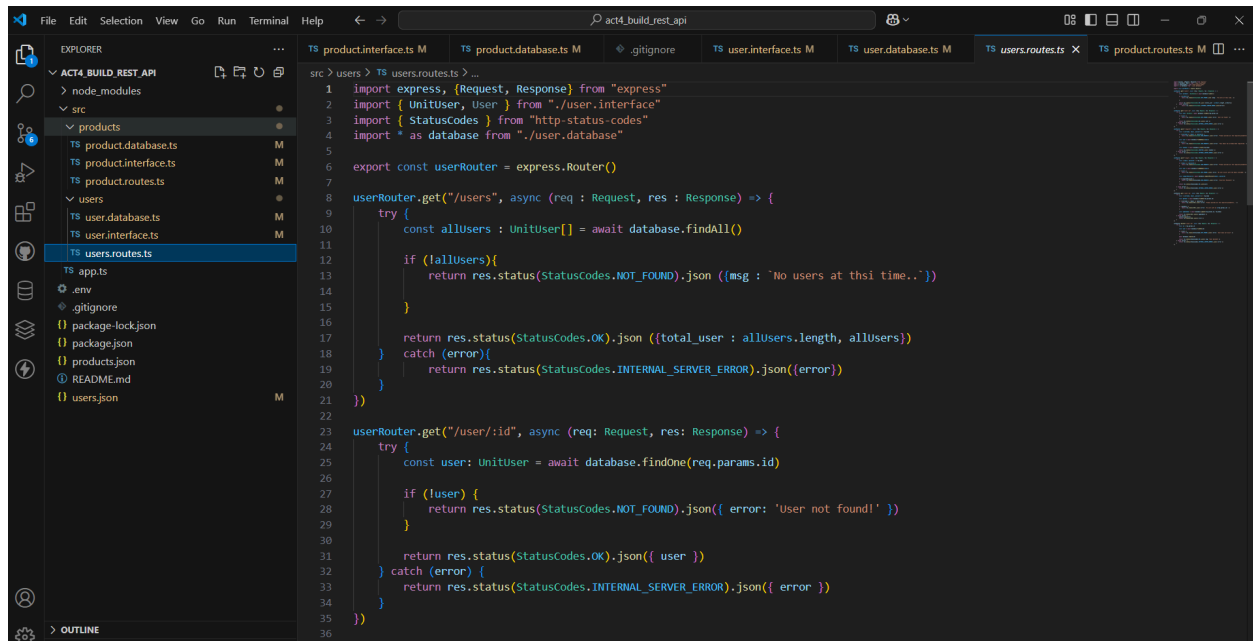
- Find the user by ID using `await findOne(id)`.
- If the user exists, generate a new salt and hash the new password using `await bcrypt.genSalt(10)` and `await bcrypt.hash(updateValues.password, salt)`.
- Update the user's password in the `updateValues` object.
- Save the updated user using `saveUsers()`.
- Return the updated user object.

The `remove` function is an asynchronous operation that takes an `id` and returns a `Promise<null | void>`. The steps are:

- Find the user by ID using `await findOne(id)`.
- If the user exists, delete it from the `users` array using `delete users[id]`.
- Save the updated array using `saveUsers()`.

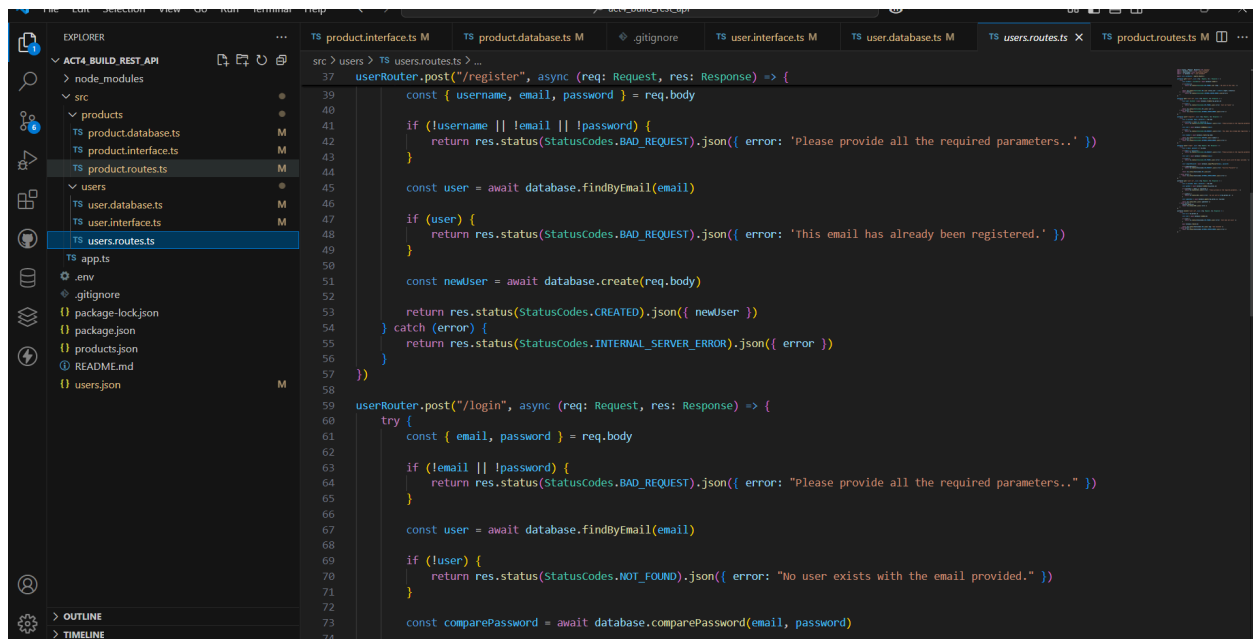
```
src > users > TS user.databases.ts > create
83 export const update = async (id: string, updateValues: User): Promise<UnitUser | null> => {
84   const userExists = await findOne(id)
85
86   if (!userExists) {
87     return null
88   }
89
90   if (updateValues.password) {
91     const salt = await bcrypt.genSalt(10)
92     const newPass = await bcrypt.hash(updateValues.password, salt)
93
94     updateValues.password = newPass
95   }
96
97   users[id] = {
98     ...userExists,
99     ...updateValues
100   }
101
102   saveUsers()
103   return users[id]
104 }
105
106 export const remove = async (id: string): Promise<null | void> => {
107   const user = await findOne(id)
108
109   if (!user) {
110     return null
111   }
112
113   delete users[id]
114
115   saveUsers()
116 }
117
118
```

Next, let all import all the required functions and modules into the routes file `./src/users.routes.ts` and populate as follows :



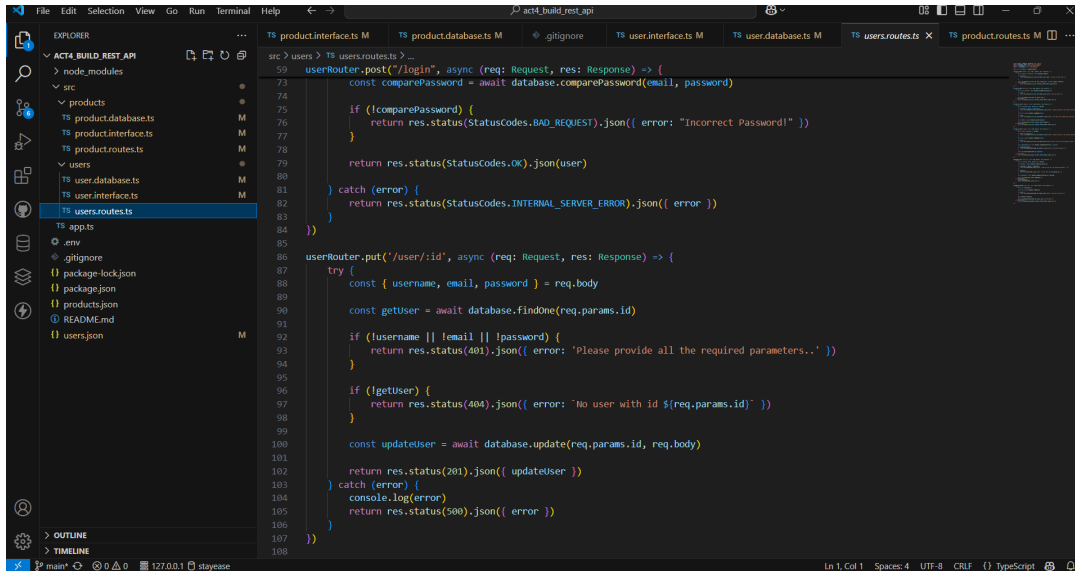
The screenshot shows the Visual Studio Code editor with the file `src > users > TS users.routes.ts` open. The Explorer sidebar on the left shows the project structure, including `src`, `products`, `users`, and `app.ts`. The main editor displays the following TypeScript code:

```
1 import express, { Request, Response } from "express"
2 import { UnitUser, User } from "../user.interface"
3 import { StatusCodes } from "http-status-codes"
4 import * as database from "../user.database"
5
6 export const userRouter = express.Router()
7
8 userRouter.get("/users", async (req : Request, res : Response) => {
9   try {
10     const allUsers : UnitUser[] = await database.findAll()
11
12     if (!allUsers){
13       return res.status(StatusCodes.NOT_FOUND).json({msg: 'No users at thsi time..'})
14     }
15
16     return res.status(StatusCodes.OK).json ({total_user : allUsers.length, allUsers})
17   } catch (error){
18     return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({error})
19   }
20 })
21
22
23 userRouter.get("/user/:id", async (req: Request, res: Response) => {
24   try {
25     const user: UnitUser = await database.findOne(req.params.id)
26
27     if (!user) {
28       return res.status(StatusCodes.NOT_FOUND).json({ error: 'User not found!' })
29     }
30
31     return res.status(StatusCodes.OK).json({ user })
32   } catch (error) {
33     return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error })
34   }
35 })
36
```



The screenshot shows the Visual Studio Code editor with the file `src > users > TS users.routes.ts` open. The Explorer sidebar on the left shows the project structure, including `src`, `products`, `users`, and `app.ts`. The main editor displays the following TypeScript code:

```
37 userRouter.post("/register", async (req: Request, res: Response) => {
38   const { username, email, password } = req.body
39
40   if (!username || !email || !password) {
41     return res.status(StatusCodes.BAD_REQUEST).json({ error: 'Please provide all the required parameters..' })
42   }
43
44   const user = await database.findByEmail(email)
45
46   if (user) {
47     return res.status(StatusCodes.BAD_REQUEST).json({ error: 'This email has already been registered.' })
48   }
49
50   const newUser = await database.create(req.body)
51
52   return res.status(StatusCodes.CREATED).json({ newUser })
53 } catch (error) {
54   return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error })
55 }
56 })
57
58
59 userRouter.post("/login", async (req: Request, res: Response) => {
60   try {
61     const { email, password } = req.body
62
63     if (!email || !password) {
64       return res.status(StatusCodes.BAD_REQUEST).json({ error: "Please provide all the required parameters.." })
65     }
66
67     const user = await database.findByEmail(email)
68
69     if (!user) {
70       return res.status(StatusCodes.NOT_FOUND).json({ error: "No user exists with the email provided." })
71     }
72
73     const comparePassword = await database.comparePassword(email, password)
74
```



The server is running port 7000

```

Microsoft Windows [Version 10.0.22631.4974]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Baggy\OneDrive\Documents\act4_build_rest_api>npm run dev

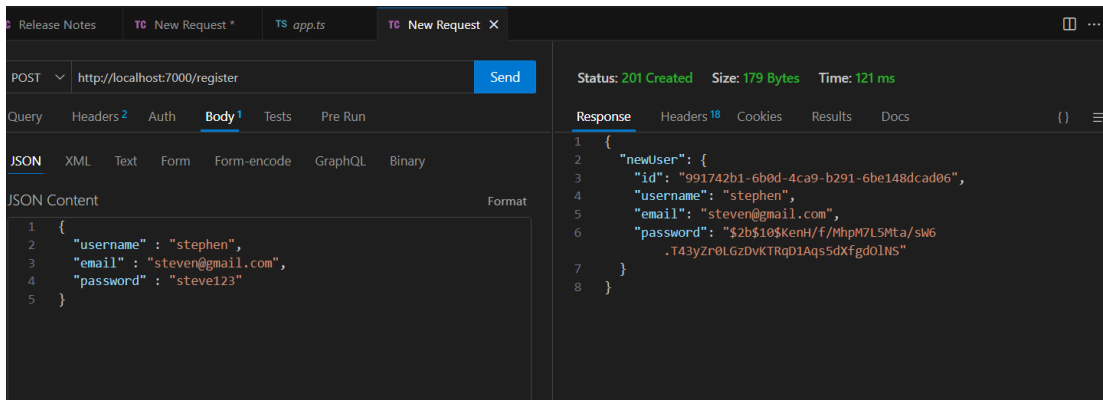
> typescript-nodejs@1.0.0 dev
> ts-node-dev --pretty --respawn ./src/app.ts

[INFO] 15:50:04 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.2, typescript ver. 4.9.5)
Server is listening on port 7000

```

So install the Thunderclient

Then , **Register users**



Login users

The screenshot shows the Postman interface with a new request tab. The request is a POST to `http://localhost:7000/login`. The body is in JSON format, containing the following content:

```
1 {
2   "email" : "stephen@gmail.com",
3   "password" : "steve123"
4 }
5
```

The response status is **200 OK**, with a size of **168 Bytes** and a time of **107 ms**. The response body is in JSON format, containing the following content:

```
1 {
2   "id": "b943cce5-873e-40ca-9e4b-3105cc90a71c",
3   "username": "stephen",
4   "email": "stephen@gmail.com",
5   "password": "$2b$10$Em6kTlHRY9DFAbLpajrIp.owKk.QEANHgT3/JPC4jeV23dBMFLR7W"
6 }
```

Get all users

The screenshot shows the Postman interface with a new request tab. The request is a GET to `http://localhost:7000/users`. The response status is **200 OK**, with a size of **366 Bytes** and a time of **6 ms**. The response body is in JSON format, containing the following content:

```
1 {
2   "total_user": 2,
3   "allUsers": [
4     {
5       "id": "b943cce5-873e-40ca-9e4b-3105cc90a71c",
6       "username": "stephen",
7       "email": "stephen@gmail.com",
8       "password": "$2b$10$Em6kTlHRY9DFAbLpajrIp.owKk.QEANHgT3/JPC4jeV23dBMFLR7W"
9     },
10    ]
11 }
```

Get a single user

The screenshot shows the Postman interface for a GET request. The URL is `http://localhost:7000/user/b943cce5-873e-40ca-9e4b-3105cc90a71c`. The response status is 200 OK, with a size of 177 Bytes and a time of 5 ms. The response body is a JSON object representing a user.

```
1 {
2   "user": {
3     "id": "b943cce5-873e-40ca-9e4b-3105cc90a71c",
4     "username": "stephen",
5     "email": "stephen@gmail.com",
6     "password": "$2b$10$Em6kTlHRY9DFAbLpajrIp.owKk.QEANHgT3/JPC4jeV23dBWFLR7W"
7   }
8 }
```

Update a user

The screenshot shows the Postman interface for a GET request. The URL is `http://localhost:7000/user/b943cce5-873e-40ca-9e4b-3105cc90a71c`. The response status is 200 OK, with a size of 177 Bytes and a time of 5 ms. The response body is a JSON object representing a user. The left sidebar shows a list of recent requests, including GET and POST requests to various endpoints.

```
1 {
2   "user": {
3     "id": "b943cce5-873e-40ca-9e4b-3105cc90a71c",
4     "username": "stephen",
5     "email": "stephen@gmail.com",
6     "password": "$2b$10$Em6kTlHRY9DFAbLpajrIp.owKk.QEANHgT3/JPC4jeV23dBWFLR7W"
7   }
8 }
```


Delete user

The screenshot shows a REST client interface. The top bar has a 'New Request' button and a dropdown menu. Below it, the 'Activity' tab is selected, showing a list of recent requests. The 'Query' tab is active, displaying the details of a DELETE request to `http://localhost:7000/user/b943cce5-873e-40ca...`. The request status is '200 OK', size is '22 Bytes', and time is '4 ms'. The response body is a JSON object: `{ "msg": "User deleted" }`.

Activity

- DEL** localhost:7000/user/b943cce5-873e-40ca... just now
- GET** localhost:7000/user/b943cce5-873e-40ca... just now
- GET** localhost:7000/user just now
- GET** localhost:7000/users just now
- POST** localhost:7000/login just now
- POST** localhost:7000/register 13 hours ago

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter	value
-----------	-------

Status: 200 OK Size: 22 Bytes Time: 4 ms

Response Headers 18 Cookies Results Docs {}

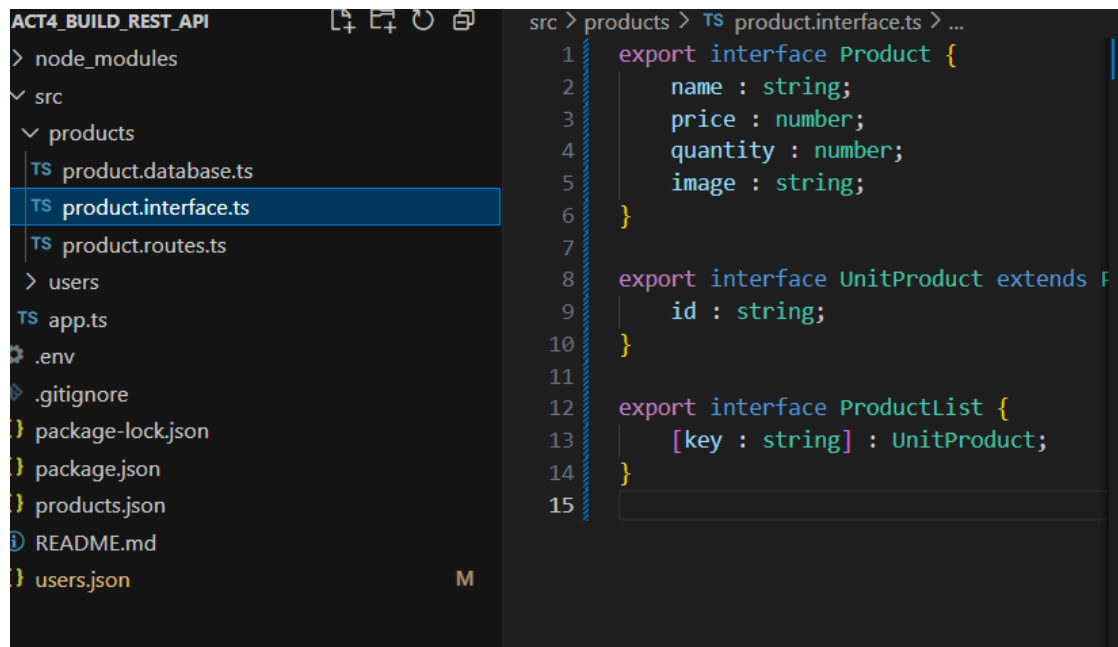
```
1 {
2   "msg": "User deleted"
3 }
```

Users-data-storage-file :

The screenshot shows a code editor with a file explorer on the left. The file explorer shows a project structure with a `users.json` file selected. The `users.json` file contains a JSON array with one user object.

```
1 {
2   "991742b1-6b0d-4ca9-b291-6be148dcad06": {
3     "id": "991742b1-6b0d-4ca9-b291-6be148dcad06",
4     "username": "stephen",
5     "email": "steven@gmail.com",
6     "password": "$2b$10$KenH/f/MhpM7L5Mta/sM6.T43yZr0LGzDVKTRqD1Aq55dXfgd0lNS"
7   }
8 }
9
10
```

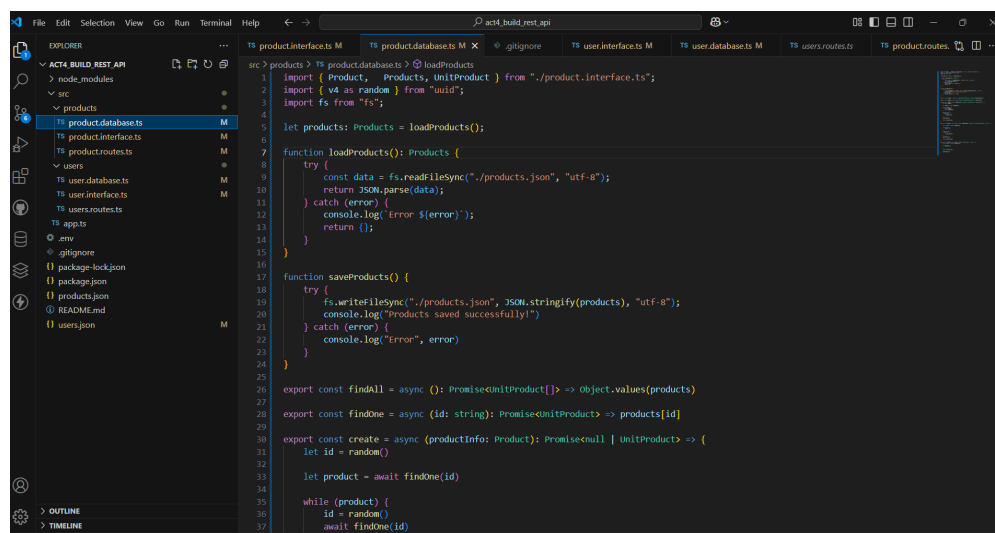
So here the products like or same in the users just create this file
./src/product.interface.ts



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure for 'ACT4_BUILD_REST_API'. The 'src' directory is expanded, showing 'products' and 'users'. Under 'products', the file 'product.interface.ts' is selected. The main editor area shows the content of this file:

```
src > products > TS product.interface.ts > ...
1  export interface Product {
2      name : string;
3      price : number;
4      quantity : number;
5      image : string;
6  }
7
8  export interface UnitProduct extends Product {
9      id : string;
10 }
11
12 export interface ProductList {
13     [key : string] : UnitProduct;
14 }
15
```

Next, just like in the ./src/users.database.ts file, same in the first step populate the ./src/products.database.ts with a similar logic. Then codes follow:



The screenshot shows the VS Code interface with the file 'product.database.ts' open in the editor. The Explorer sidebar on the left shows the project structure, with 'product.database.ts' selected under the 'products' directory. The main editor area displays the following code:

```
src > products > TS product.database.ts M
1  import { Product, Products, UnitProduct } from './product.interface.ts';
2  import { v4 as random } from 'uuid';
3  import fs from 'fs';
4
5  let products: Products = loadProducts();
6
7  function loadProducts(): Products {
8      try {
9          const data = fs.readFileSync('./products.json', 'utf-8');
10         return JSON.parse(data);
11     } catch (error) {
12         console.log('Error: ', error);
13         return [];
14     }
15 }
16
17 function saveProducts() {
18     try {
19         fs.writeFileSync('./products.json', JSON.stringify(products), 'utf-8');
20         console.log('Products saved successfully!');
21     } catch (error) {
22         console.log('Error: ', error);
23     }
24 }
25
26 export const findAll = async (): Promise<UnitProduct[]> => Object.values(products)
27
28 export const findOne = async (id: string): Promise<UnitProduct> => products[id]
29
30 export const create = async (productInfo: Product): Promise<null | UnitProduct> => {
31     let id = random();
32
33     let product = await findOne(id)
34
35     while (product) {
36         id = random();
37         await findOne(id)
38     }
39 }
```

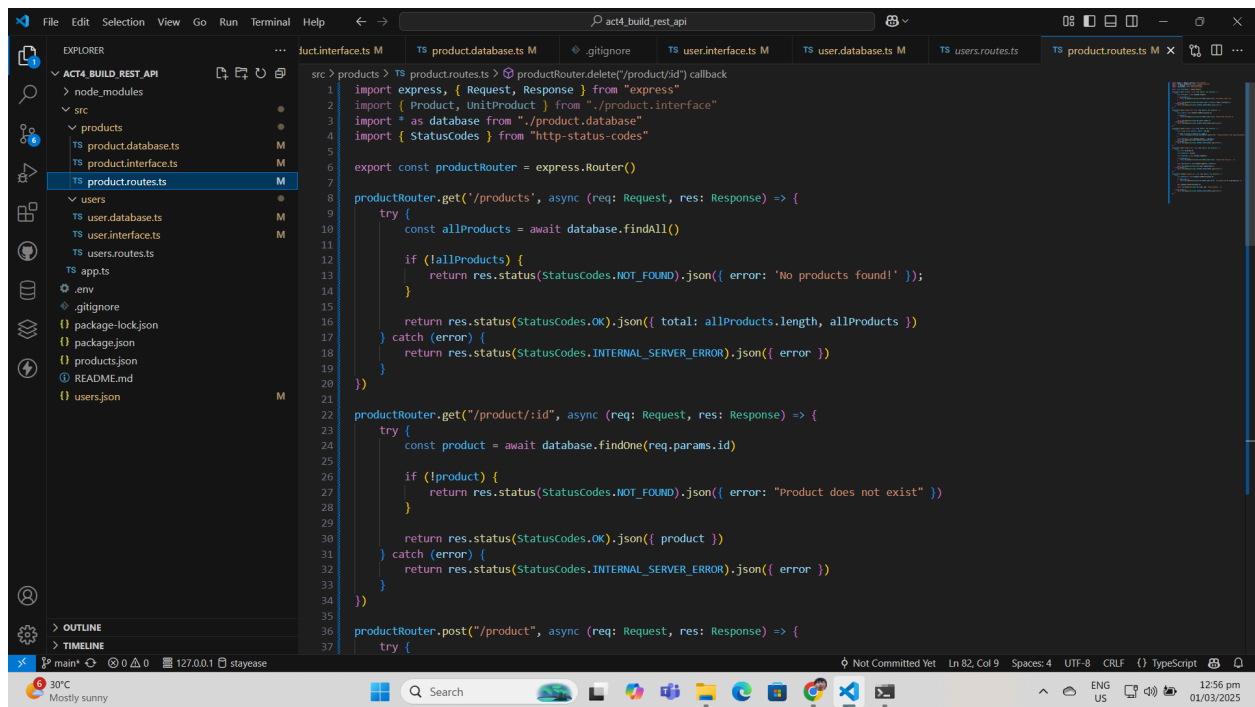
This screenshot shows the VS Code editor with the file explorer on the left displaying the project structure for 'ACT4_BUILD_REST_API'. The main editor window shows the 'product.database.ts' file with the following TypeScript code:

```
src > products > TS product.database.ts > loadProducts
30 export const create = async (productInfo: Product): Promise<null | UnitProduct> => {
39
40   products[id] = {
41     id: id,
42     ...productInfo
43   }
44
45   saveProducts()
46
47   return products[id]
48 }
49
50 export const update = async (id: string, updateValues: Product): Promise<UnitProduct | null> => {
51
52   const product = await findOne(id)
53
54   if (!product) {
55     return null
56   }
57
58   products[id] = {
59     id,
60     ...updateValues
61   };
62
63   saveProducts()
64
65   return products[id]
66 }
67
68 export const remove = async (id: string): Promise<null | void> => {
69   const product = await findOne(id);
70
71   if (!product) {
72     return null
73   }
74 }
```

This screenshot shows the VS Code editor with the file explorer on the left displaying the project structure for 'ACT4_BUILD_REST_API'. The main editor window shows the 'product.database.ts' file with the following TypeScript code:

```
src > products > TS product.database.ts > loadProducts
50 export const update = async (id: string, updateValues: Product): Promise<UnitProduct | null> => {
51   const product = await findOne(id)
52
53   if (!product) {
54     return null
55   }
56
57   products[id] = {
58     id,
59     ...updateValues
60   };
61
62   saveProducts()
63
64   return products[id]
65 }
66
67 export const remove = async (id: string): Promise<null | void> => {
68   const product = await findOne(id);
69
70   if (!product) {
71     return null
72   }
73
74   delete products[id]
75
76   saveProducts()
77
78 }
79
80 }
```

So here is the routes implement the the routes for our products
Populate the ./src/products.routes.ts file with the following code :



The screenshot shows a Visual Studio Code editor window with the file explorer on the left and the code editor in the center. The file explorer shows the project structure for 'ACT4_BUILD_REST_API', with the 'src' directory expanded and 'products.routes.ts' selected. The code editor displays the following TypeScript code:

```
src > products > TS product.routes.ts > productRouter.delete("/product/:id") callback
1 import express, { Request, Response } from "express"
2 import { Product, UnitProduct } from "../product.interface"
3 import { database } from "../product.database"
4 import { StatusCodes } from "http-status-codes"
5
6 export const productRouter = express.Router()
7
8 productRouter.get("/products", async (req: Request, res: Response) => {
9   try {
10     const allProducts = await database.findAll()
11
12     if (!allProducts) {
13       return res.status(StatusCodes.NOT_FOUND).json({ error: 'No products found!' });
14     }
15
16     return res.status(StatusCodes.OK).json({ total: allProducts.length, allProducts })
17   } catch (error) {
18     return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error })
19   }
20 })
21
22 productRouter.get("/product/:id", async (req: Request, res: Response) => {
23   try {
24     const product = await database.findOne(req.params.id)
25
26     if (!product) {
27       return res.status(StatusCodes.NOT_FOUND).json({ error: "Product does not exist" })
28     }
29
30     return res.status(StatusCodes.OK).json({ product })
31   } catch (error) {
32     return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error })
33   }
34 })
35
36 productRouter.post("/product", async (req: Request, res: Response) => {
37   try {
```

The status bar at the bottom indicates the file is 'Not Committed Yet' at line 82, column 9, with 4 spaces and UTF-8 encoding. The system tray shows a temperature of 30°C and the date 01/03/2025.

The screenshot shows the VS Code editor with the file explorer on the left. The project structure is as follows:

- ACT4_BUILD_REST_API
 - node_modules
 - src
 - products
 - product.database.ts
 - product.interface.ts
 - product.routes.ts (selected)
 - users
 - user.database.ts
 - user.interface.ts
 - users.routes.ts
 - app.ts
 - .env
 - .gitignore
 - package-lock.json
 - package.json
 - products.json
 - README.md
 - users.json

The main editor displays the code for `product.routes.ts`. The code implements the following routes:

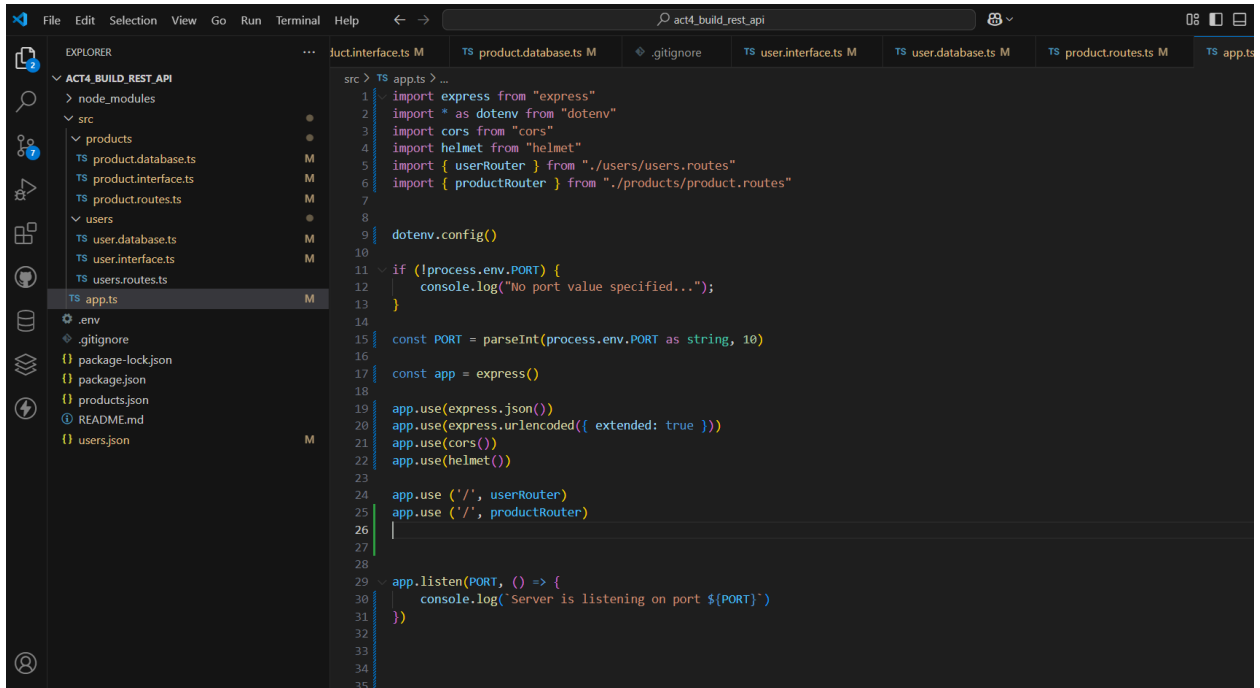
- `productRouter.post("/product", async (req: Request, res: Response) => {`
 - Try block:
 - Extract `name, price, quantity, image` from `req.body`.
 - Validation: `if (!name || !price || !quantity || !image) { return res.status(StatusCodes.BAD_REQUEST).json({ error: 'Please provide all the required parameters..' }) }`
 - Creation: `const newProduct = await database.create({ ...req.body })`
 - Response: `return res.status(StatusCodes.CREATED).json(newProduct)`
 - Catch block: `catch (error) { return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error }) }`

- `productRouter.put("/product/:id", async (req: Request, res: Response) => {`
- Try block:
 - Extract `id` from `req.params.id`.
 - Update: `const newProduct = req.body`
 - Find: `const findProduct = await database.findOne(id)`
 - Validation: `if (!findProduct) { return res.status(StatusCodes.NOT_FOUND).json({ error: 'Product does not exist..' }) }`
 - Update: `const updateProduct = await database.update(id, newProduct)`
 - Response: `return res.status(StatusCodes.OK).json({ updateProduct })`
- Catch block: `catch (error) { return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error }) }`
- `productRouter.delete("/product/:id", async (req: Request, res: Response) => {` (partially visible)

The screenshot shows the VS Code editor with the file explorer on the left. The project structure is the same as in the previous image. The main editor displays the code for `product.routes.ts`, showing the implementation of the `productRouter.delete` route:

- `productRouter.delete("/product/:id", async (req: Request, res: Response) => {`
 - Try block:
 - Find product: `const getProduct = await database.findOne(req.params.id)`
 - Validation: `if (!getProduct) { return res.status(StatusCodes.NOT_FOUND).json({ error: 'No product with ID ${req.params.id}' }) }`
 - Deletion: `await database.remove(req.params.id)`
 - Response: `return res.status(StatusCodes.OK).json({ msg: 'Product deleted..' })`
 - Catch block: `catch (error) { return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error }) }`

Then , this is the updated app.ts for products and users

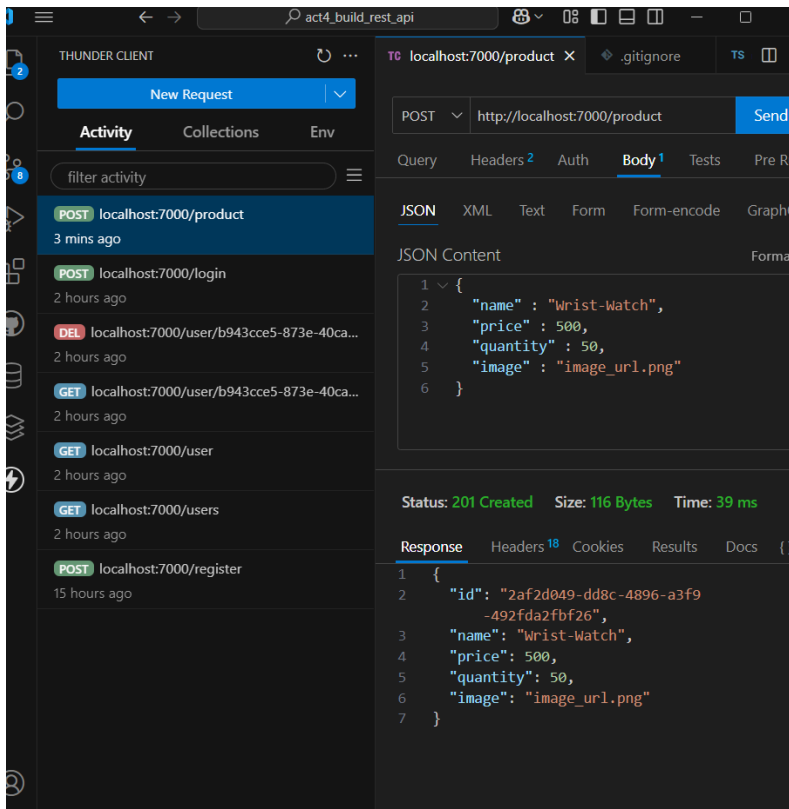


The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure for 'ACT4_BUILD_REST_API'. The code editor displays the 'app.ts' file, which is a TypeScript file for an Express.js application. The code includes imports for 'express', 'dotenv', 'cors', 'helmet', 'userRouter', and 'productRouter'. It sets up the application with 'dotenv.config()', 'cors', 'helmet', and the routers. The application listens on a specified port (defaulting to 3000) and logs the server status.

```
src > TS app.ts > ...
1 import express from "express"
2 import * as dotenv from "dotenv"
3 import cors from "cors"
4 import helmet from "helmet"
5 import { userRouter } from "../users/users.routes"
6 import { productRouter } from "../products/product.routes"
7
8
9 dotenv.config()
10
11 if (!process.env.PORT) {
12   console.log("No port value specified...");
13 }
14
15 const PORT = parseInt(process.env.PORT as string, 10)
16
17 const app = express()
18
19 app.use(express.json())
20 app.use(express.urlencoded({ extended: true }))
21 app.use(cors())
22 app.use(helmet())
23
24 app.use('/', userRouter)
25 app.use('/', productRouter)
26
27
28
29 app.listen(PORT, () => {
30   console.log(`Server is listening on port ${PORT}`)
31 })
32
33
34
35
```

Then, let's have a test

Create product



The screenshot shows the Thunder Client interface. The 'Activity' tab is selected, showing a list of requests. The first request is a POST to 'localhost:7000/product' made 3 minutes ago. The 'Body' tab is selected for this request, showing the JSON content: { "name": "Wrist-Watch", "price": 500, "quantity": 50, "image": "image_url.png" }. The status bar indicates a 201 Created status, 116 Bytes size, and 39 ms time.

Activity

- POST localhost:7000/product 3 mins ago
- POST localhost:7000/login 2 hours ago
- DEL localhost:7000/user/b943cce5-873e-40ca... 2 hours ago
- GET localhost:7000/user/b943cce5-873e-40ca... 2 hours ago
- GET localhost:7000/user 2 hours ago
- GET localhost:7000/users 2 hours ago
- POST localhost:7000/register 15 hours ago

POST http://localhost:7000/product

Query Headers Auth Body Tests Pre Ru

JSON XML Text Form Form-encode GraphC

JSON Content

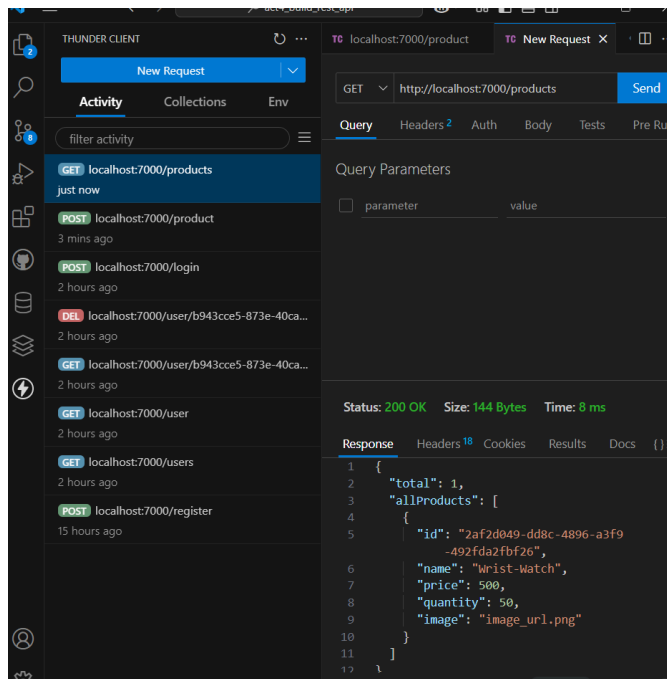
```
1 {
2   "name": "Wrist-Watch",
3   "price": 500,
4   "quantity": 50,
5   "image": "image_url.png"
6 }
```

Status: 201 Created Size: 116 Bytes Time: 39 ms

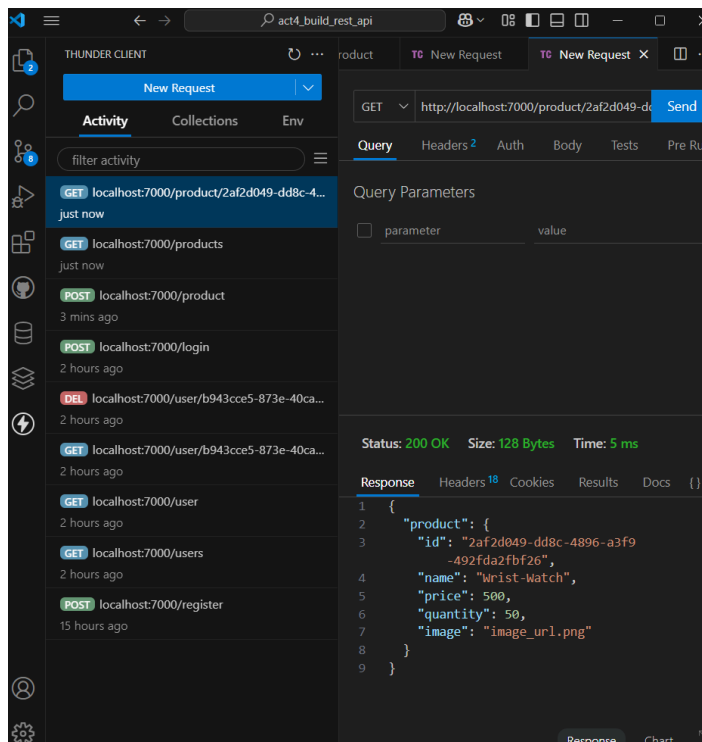
Response Headers Cookies Results Docs {}

```
1 {
2   "id": "2af2d049-dd8c-4896-a3f9-492fda2fbf26",
3   "name": "Wrist-Watch",
4   "price": 500,
5   "quantity": 50,
6   "image": "image_url.png"
7 }
```

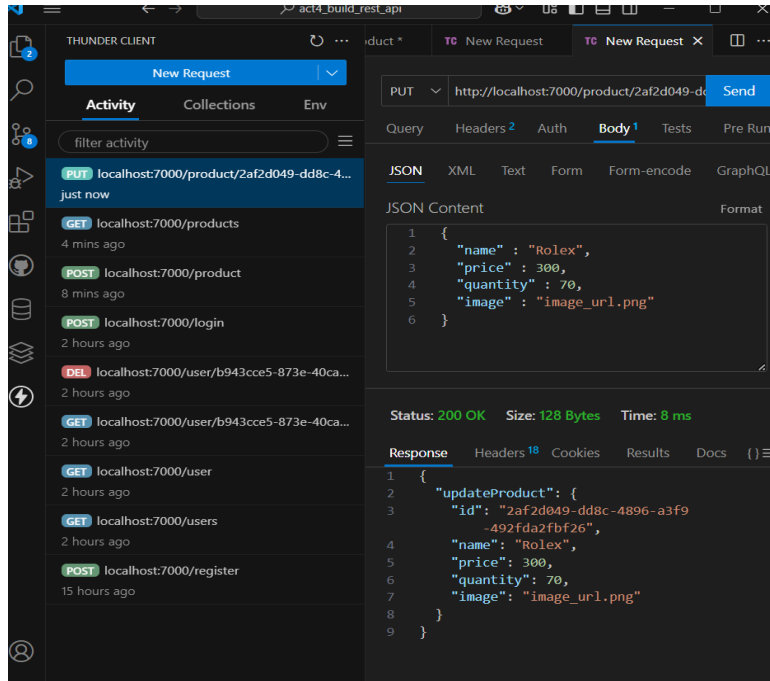
All products



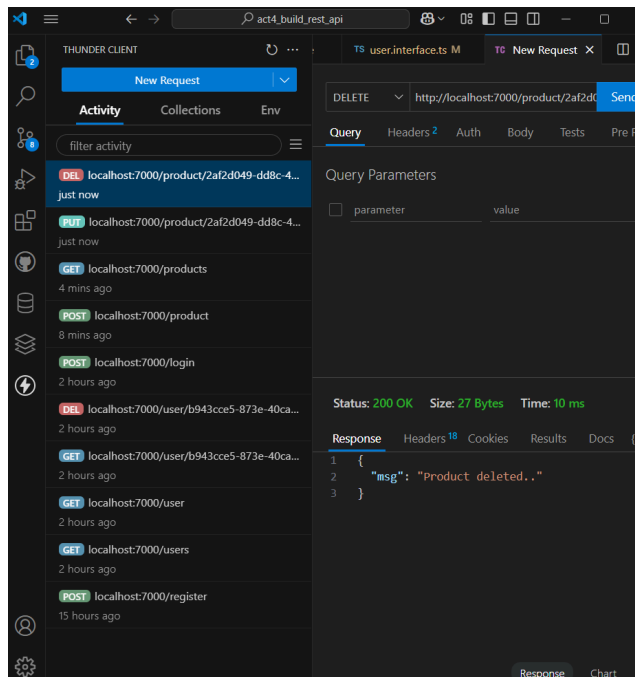
Single product



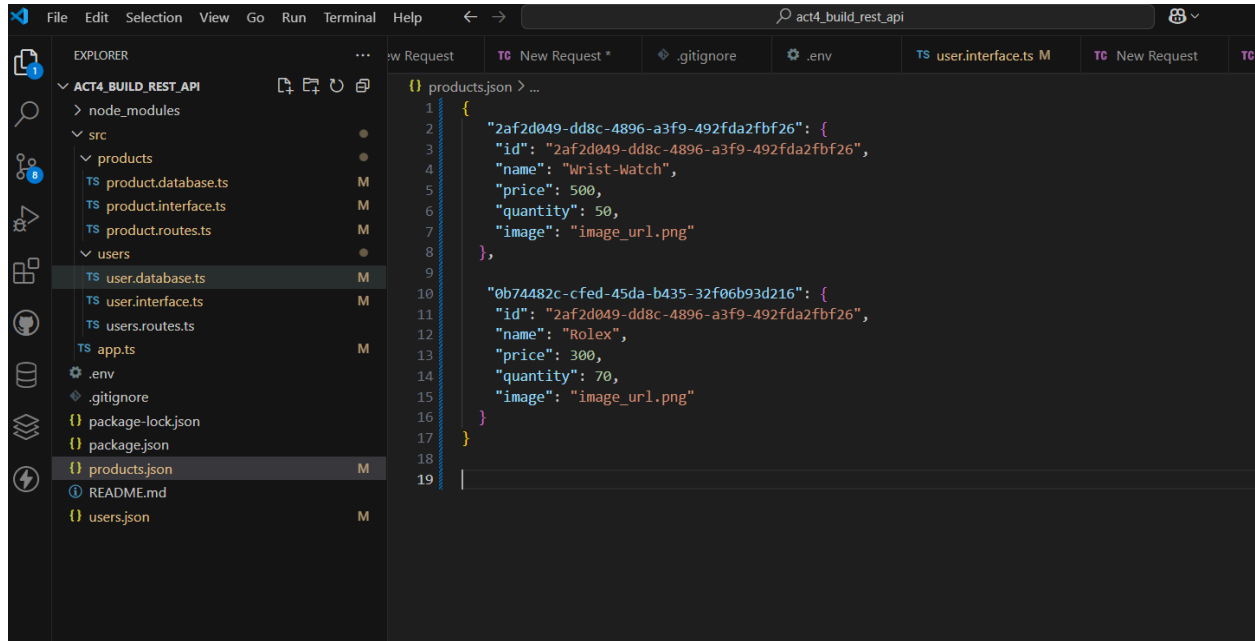
Update product



Delete product



Here is the new product added will be appended to the products.json file



```
1 {
2   "2af2d049-dd8c-4896-a3f9-492fda2fbf26": {
3     "id": "2af2d049-dd8c-4896-a3f9-492fda2fbf26",
4     "name": "Wrist-Watch",
5     "price": 500,
6     "quantity": 50,
7     "image": "image_url.png"
8   },
9
10  "0b74482c-cfed-45da-b435-32f06b93d216": {
11    "id": "2af2d049-dd8c-4896-a3f9-492fda2fbf26",
12    "name": "Rolex",
13    "price": 300,
14    "quantity": 70,
15    "image": "image_url.png"
16  }
17 }
18
19 |
```

https://github.com/Eldrintrapa23/act4_build_rest_api.git

